UNCLASSIFIED

Defense Technical Information Center
Compilation Part Notice

ADP021388

TITLE: Joint Synthetic Battlespace [JSB] Technology and Infrastructure Consideration

DISTRIBUTION: Approved for public release, distribution unlimited

This paper is part of the following report:

TITLE: International Conference on Integration of Knowledge Intensive Multi-Agent Systems. KIMAS '03: Modeling, Exploration, and Engineering Held in Cambridge, MA on 30 September-October 4, 2003

To order the complete compilation report, use: ADA441198

The component part is provided here to allow users access to individually authored sections of proceedings, annals, symposia, etc. However, the component should be considered within the context of the overall compilation report and not as a stand-alone technical report.

The following component part numbers comprise the compilation report:
ADP021346 thru ADP021468

UNCLASSIFIED

Joint Synthetic Battlespace (JSB) Technology and Infrastructure Consideration

David Kwak, PhD, ESC/CX-MITRE, 781-271-6431, dkwak@mitre.org
Lt. Col. Emily Andrew, ESC/CX, 781-377-6421, emily.andrew@hanscom.af.mil
Capt. Jack Murtha, ESC/CX, john.murtha@hanscom.af.mil
Capt. Lucas Flanagan, ESC/CX, lucas.flanagan@hanscom.af.mil
Lt. Denise Herrera, ESC/CX, denise.herrera@hanscom.af.mil
Lt. Don Brown, ESC/CX, don.brown@hanscom.af.mil
15 Eglin St, Hanscom AFB, MA, 01731, USA

Abstract— Creating all components from scratch is impractical for the Joint Synthetic Battlespace (JSB) due to the huge scope that JSB needs to cover. Spanning training, acquisition, test and evaluation, and research and development, JSB supports the DoD from the detailed engineering level, entity level, mission level, to the operational and strategic levels, while supporting community users. Thus, the JSB must leverage all existing investments to the fullest extent possible. It is expected that existing simulations cannot cover some of the JSB space even though all the existing components are fully integrated under the JSB infrastructure. The missing pieces will need to be created to achieve the goal and vision of the JSB. The JSB will therefore be a combination of legacy simulations and the newly created JSB components in future. This paper discusses the JSB component integration framework that facilitates integration of the existing legacy simulation systems as well as one of the new JSB components, the JSB Common Synthetic Environment (JSB CSE). It also presents the on-going High Performance Computing (HPC) applications of the JSB as its computing infrastructure.

1. INTRODUCTION

In USAF, there is a newly emerging need to support development, acquisition, and deployment of Task Forces. Global Strike Task Force (GSTF), one of the USAF Task Forces, requires fully integrating all existing and newly developed systems to achieve the new level of synergistic capability. GSTF, thus, effectively becomes a seamlessly integrated system of weapon/C4ISR systems. However, the currently available systems have been essentially developed in a stove-piped fashion, and now they are obstacles preventing the full vision of GSTF. Therefore, newly developed systems have to be explicitly conceptualized, designed and constructed as a part of the bigger GSTF context, and the existing systems have to be integrated in the GSTF. GSTF does not exist now, but it can be created with computer simulation.

KIMAS 2003, October 1-3, 2003, Boston, MA, USA.
Copyright 0-7803-7958-6/03/$17.00 © 2003 IEEE.

Joint Synthetic Battlespace was the simulation chosen for GSTF, and it will become one of the enabling technologies that support development of new systems and migration of the existing systems toward full realization of the GSTF vision.

Therefore, the JSB's mission allows for integration of both legacy and newly developed simulations. Current simulations suffer from effects of stove-piped development, and they do not interoperate well with other simulations. Most of existing simulations have been created to support a narrow specification. The simulation world is significantly fragmented and, as it is, it lacks the capability to support the highly integrated and complex GSTF. It is commonly known that any integration efforts are greatly hampered by implicit and explicit design assumptions and implementation limitations associated with the existing simulation systems.

Joint Synthetic Battlespace (JSB), thus, does not suggest yet another simulation. It is a common simulation architecture and core services, and it is capable of integrating current and future simulations while supporting USAF needs from acquisition, research and development, training, and mission planning to mission rehearsal. This paper briefly describes the JSB vision, and the conceptual instantiation of the JSB. Then the discussion of the JSB Integration Framework follows. The JSB Beowulf cluster and JSB visual requirements will be also discussed.

2. JOINT SYNTHETIC BATTLESPACE VISION

The Joint Synthetic Battlespace is an on-demand, integrated environment and operational battlespace, able to selectively accommodate different functional applications at varying levels of detail using common components. While the detail of the CONOPS of JSB is available as a separate document [1], four areas are briefly presented here:

A New Problem Space Is Emerging

Traditional military systems have been designed for a narrow application domain, as is the acquisition process. However, future weapon systems, such MC2C (Multi Command and Control Constellation) and JSTF (Joint

Strike Task Forces), are different. They are essentially a system of systems with an unprecedented complexity because of the combinatorial complexity of behavioral interactions among them [2]. Technical and operational testing as well as evaluations performed, as a part of acquisition process have to be matched the level of sophistication accordingly. The cost of the future systems will be much higher than now. Thus, a sophisticated simulation-based acquisition (SBA) process becomes critical, which is capable of supporting full life cycle of all acquisition process and beyond. This is the vision of JSB.

90% of solutions are out there.

JSB will not be built from scratch, but will be largely composed of existing simulations. ESC's (Electronic Systems Center) preliminary analysis shows that 90% of the solutions are out there waiting for construction of the JSB. Figure 1 depicts the Command Enterprise Integration System (CEIS) Government-Industry Partnership that is directly applicable to JSB.

They are not interoperable.

Although many relevant simulations already exist, they do not generally interoperate because they were created to serve their own needs. Incompatible requirements, architecture, technologies/standards (some of which are homegrown) are the main obstacles. Ad hoc integration approaches have been tried, but success has been limited. A new and systematic approach that is able to take advantage of full potentials of the existing investments while meeting the new vision of JSB is a must.

They need a common environment.

True interoperation is more than just having a common interface scheme. A common interface essentially means a common data exchange protocol which only resolves the syntactic differences among them. Each simulation comes with its own baggage: different objectives, assumptions, limitations, resolutions and fidelities. These hidden (semantic) differences bring in a real challenge. It is necessary to create an environment where they can resolve those semantic differences. As a common understanding between humans, having different ideas is a pre-requisite for meaningful communication. JSB needs a common environment through which multiple simulations can communicate. A correlated multi-spectrum environment, for example, allows for consistent operations of multiple sensors in simulation. Another important area for JSB is creation of an integration frame for JSB through a common contextual understanding, which is described in this paper.

3. JSB OVERARCHING FRAMEWORK

JSB will not be a single, monolithic simulation system that satisfies all of the JSB requirements described above. If it were, the JSB would be an enormous simulation system whose size would easily draft the size of current joint generation simulations such JSIMS (Joint Simulation System). JSB needs to support both acquisition and operational users, whereas JSIMS is designed to support military training only. In fact, training is merely a small subset of military operational usages. It is commonly understood that the current state of art simulation technology will not allow for a single monolithic system that meets all the requirements of the JSB.

A JSB will be, thus, instantiated from a common JSB reference architecture, and each instance will be able to meet its specific needs, such as military training, acquisition decision, concept development, etc. That is, there will be no single JSB simulation system, but multiple JSB instances constructed as needed. This insatiable simulation system, which is an innovative approach, will truly make JSB different from existing DOD simulations, and the full extent of JSB requirements will be fully satisfied by the JSB instances collectively.

As shown in the figure, a JSB instance is a product of JSB system engineering process. The JSB system engineering process is tightly coupled with the JSB executable architecture. The JSB Executable Architecture allows for early exploration, investigation, and design of internal structures and controls flow of the target JSB instance without actual internal simulation components. Inputting

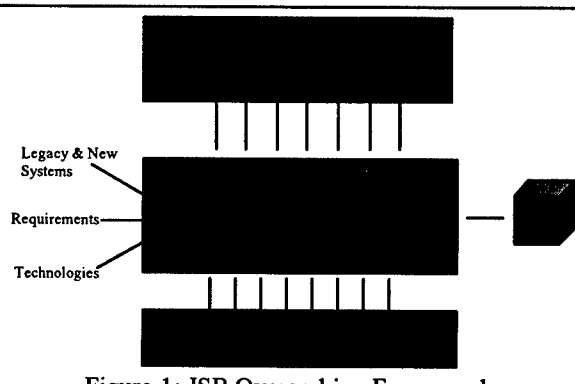


Figure 1: JSB Overarching Framework

raw materials of the above JSB process are legacy and new simulation systems, users' requirements, and existing and new technologies. The JSB Integration Framework provides an actual means to put together the JSB internal components following the overall structure and the control flow captured in the JSB executable architecture. Finally, the JSB IDE (Integrated Digital Environment) facilitates storing, retrieving, moving around data/artifacts/knowledge generated, used, and acquired by internal processes of the above Overall JSB Framework. That is, the JSB IDE becomes an ultimate repository and tool for JSB system configuration management, documentations, program management, collaboration, distributed manufacturing of JSB instances.

A series of documents required to create a specific JSB instance is supported by the JSB in the central knowledge

repository of the IDE. That is, a user requirement document, technical document, program management document will be generated from the central knowledge repository. IDE's multiple view generation capability supports semi-automated creation of multiple documents from a single central knowledge store in the IDE. After all, those documents are multiple views (i.e., manifestations) of a common knowledge that requires for an instantiation of a specific JSB instance.

Presently, human authors manually create the documents. Collecting necessary knowledge by parsing the existing document, adding additional knowledge, and assembling them in a specific format, a target document is created. For example, JSB CONOPS (Concept of Operations) is one of the JSB documents, and is manually created by a group of people. The JSB IRD (Interim Requirement Document) is subsequently created by another group of people. JSB CONOPS, the baseline document is manually parsed to form a knowledge base, and new knowledge specific to JSB IRD is added to the knowledge. If the first group of the people, who created the JSB IRD, do not involve in the JSB IRD generation process, a large portion of the valuable knowledge collected and created during JSB CONOPS creation process is lost. The knowledge captured in a form of a JSB CONOPS document is really limited compared to the knowledge actually captured by the first documentation group. In reality, most of the knowledge is simply stored in the brains of the humans participating in the documentation process. After the CONOPS generation process, the knowledge captured in the human brains has little chance to be carried over to the subsequent JSB IRD generation process unless the same people are involved. Although the original group has been involved, the temporal gap between the two document generations undermines the effectiveness of communication. The current form of the CONOPS is, after all, one of the best known means to capture the knowledge; however, it is certainly a narrow channel for transferring knowledge. Again, there exists inefficiency of restoring the knowledge captured in the CONOPS by reading the CONOPS document, reassembling the knowledge base in the human brains, and creating the new JSB IRD documentation.

During the above parsing and reassembly process, new knowledge is obviously added. As new documentation is created following the process of creation of a JSB instance, the size of accumulated knowledge is increased as well as the size of the documents. Thus, the inefficiency of unpacking and packing knowledge from one document to the next is also increased. This process would continue until a complete set of documentation becomes available to build a JSB instance.

This above conventional manual knowledge accumulation process is manually intensive. Even so, the current process focuses on manual creation of written documentations partly because it is a familiar form of knowledge capturing since the invention of writing and partly because there is no other alternative. It is also true that this manual process has been focused on standardization on manifestation of knowledge on papers, rather than the standardizing of storing the source knowledge. Thus, reusing the captured knowledge at the source level could not have been achieved at all. In reality, many documents have been historically created to simply satisfy a documentation requirement of a given process rather than to facilitate knowledge transfer. There is little chance to expect a significant change in the near future to having a truly re-usable knowledge representation among the document generation groups.

There is an emerging technology in this area, and the JSB IDE is such a technology. It supports standardization of knowledge representation, and provides tools and utilities for capturing, manipulating and extracting various views and formats. Once knowledge is electronically captured in the tool rich environment, it becomes much more powerful than any textual form of knowledge, which is static and not-easily transformable. Unpacking and packing knowledge stored in conventional documents are undeniably time consuming, but the JSB IDE will allow for a direct management of knowledge, and automatically maintain the evolution of stored knowledge.

When the above IDE knowledge approach is fully adopted, JSB instance creation will be greatly facilitated. JSB system engineers will directly interact with the knowledge in a form that they need rather than passive documents written on paper. Moreover, the JSB IDE will support a backward compatibility so that the knowledge in the central JSB IDE repository will support semi-automatic creation of paper documentations mandated by DOD.

The captured knowledge in the JSB IDE does not just facilitate creation of mandated acquisition documentations, but also directly supports other functionalities that required supporting instantiations of JSB instances. The JSB IDE's multiple view of the central knowledge repository is capable of generating other documents and information such as engineering, managerial, and financial aspect of JSB. Again, all of the knowledge is captured in the central repository, and evolves together. Thus, although vastly different documents and artifacts are generated from the central repository, the JSB IDE will automatically maintain consistency across all of the documents that are generated from the IDE. Traditionally, maintaining consistencies across totally disjoint groups of people such as financing, management, engineering, etc has been a difficult and time-consuming task. The JSB IDE will vastly improve this problem.

4. JSB INTEGRATION FRAMEWORK (JSB IF)

The JSB Integration framework permits JSB to have true plug and play architecture. The internal structure of the JSB Integration framework is shown in Figure 2.

The ultimate goal of JSB is to rapidly instantiate JSB instances by integrating existing (i.e., legacy) simulation systems and models to a target JSB instance. Creation of a new component will be limited to a situation that no legacy simulation system/model supports the required functionality. Therefore, creating a JSB instance becomes a composition task rather than an ordinary production task. This concept has been metaphorically captured as Lego pieces or Ga me Machine and Game Cartridges [3].

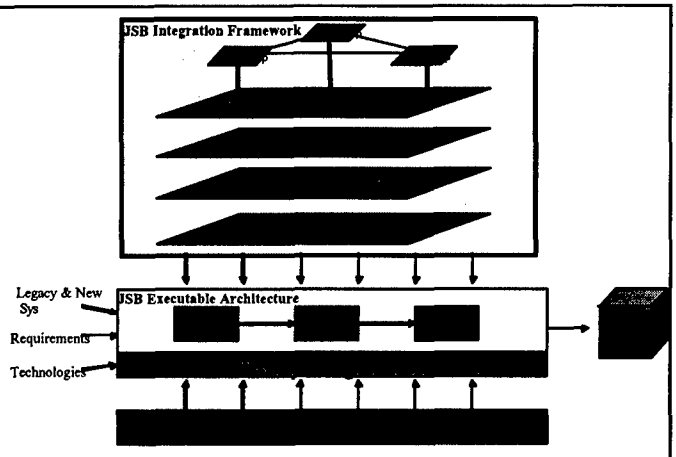


Figure 2: JSB Integration Framework is composed of computing, network, integration and plug-and-play infrastructures.

Historically, many approaches, architectures, and protocols have been introduced to achieve the Lego style Plug-and-Play capability for simulation systems, but they are fall short of the fullness of the true plug and play capability. Often a proposed plug and play scheme is too manually intensive, or the scope is too narrow to practically cover the whole simulation system composition issue while really saving significant level of efforts on creation of simulation systems. The JSB Integration framework precisely addresses these issues. The JSB Integration framework divides the plug and play problem space into four sub-spaces called: computational infrastructure, networking infrastructure, integration infrastructure, and plug and play architecture.

In this document, the focus will be given to the top two layers. The bottom two layers – JSB Computing and JSB Network infrastructures – will extensively leverage technologies developed and matured by the commercial industry. Each year, computing capabilities of computing hardware have demonstrated explosive improvements on their performances. The Moore's law often explains this phenomenon. For example, Intel-architecture based PCs have closely followed Moore's law for decades. The storage size of internal disks and storage area networks has also increased following the similar pattern.

The network infrastructure is another area that has been making remarkable progresses. In some sense, the growth rate of the network infrastructure has been surpassing the rate of Moore's law [4] predicts especially in recent years. Literally, in the last 10+ years, the entire Web industry was established and exponentially flourished while creating enormous business opportunities on the Web. Therefore, synchronizing the JSB computing and network infrastructures with the commercially available technologies is a wise tactic for the JSB Integration framework. The bottom two JSB infrastructures will automatically upgrade by huge investments from the commercial industry, and enjoy the fruits from the investments without requiring extensive DOD investments on the bottom two JSB infrastructures.

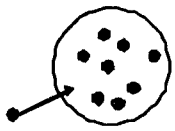
On the other hand, for the top two layers, there is little reason why the commercial sector would invest heavily in these technologies because they are quite specific to JSB. Thus, these two layers will be developed by JSB investments. Again, whenever possible, commercial technologies will be heavily leveraged for the top two layers. The JSB Integration Infrastructure (JSB I2) is uniquely designed to allow for integration of legacy simulation systems regardless of their protocol and standards (i.e., ALSP, DIS and HLA/RTI) into a JSB instance. JSB I2 also should facilitate integration of new JSB compliant simulation models, components and systems. The JSB I2 turns virtually any simulation components (legacy or new) to a JSB compliant plug and playable components. Finally, the JSB Plug and Play architecture provides a true plug and play capability of JSB compliant components. Therefore, the top two layers are unique to the JSB Integration framework, and together with the bottom two layers, a truly plug and play architecture is constructed. The details of the top two layers will be discussed in later sections.

5. COMMONLY USED INTEGRATION APPROACHES

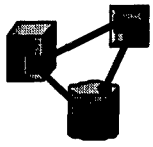
Before discussing the details of the JSB IF, three commonly used integration approaches are presented to constitute a common point of departure toward JSB IF discussion in later sections.

The first approach is *Common Interface Based Integration*. Traditional system engineering falls in the first approach. All of the internal components are precisely defined and controlled by a strict engineering process. For example, the entire interface definition of all participating components is clearly pre-defined. It is usually captured in an interface control document. Creating a complete set of static interface definitions is a rather natural process because the functions of the internal components are statically definable except small changes in later parts of the system life. Thus, the internal components can be manufactured under precise system engineering utilizing the pre-defined system/subsystem definitions and interface specifications. In the simulation world, JSIMS (Joint Simulation System) falls in this category. That is, all of the JSIMS components have to confirm the JSIMS interface specifications before integration into JSIMS. The picture depicts this fact by

making all components in a same circle shape, and the external component joining in has the same circle shape. Due to the strict traditional engineering process (i.e., all components are made to confirm the given standards), the time scale of construction of such a system tends to be long. A typical time line is about 5 years and more.



Common Interface Based Integration
- All components must confirm the single system wide standard.
- Good for a stable system integration
- Traditional system engineering mainly falls in this category.
- Ex) JSIMS, etc
- Timeline: > 5 years



Ad Hoc Interface Based Integration
- Component system are modified for integration.
- Good for supporting coarse level ad hoc system integration
- Lab style experimentation/prototyping is in this category.

Ex) MC02, etc
- Timeline: about 1 year

Middleware Based Integration
- Middleware is created as needed.
- Good for system integration without modifying components
- Most of Web/IT based integrations fall in this category.

Ex) Web based SIM, etc
- Timeline: < 1 year

Figure 3: Three Commonly Adopted Integration Approaches

Another commonly used approach is *Ad Hoc Interface Based Integration*. This approach is often adopted by a lab style experimentation/prototyping. Unlike the above approach, interfaces of the components have not been manufactured under a single pre-defined interface specification. Therefore, most of the participating components/models/systems do not interoperate as they are. Case-by-case surgical interface modifications are needed. This integration approach is usually adopted to create a temporary experimental system. MC02 (Millennium Challenge 02) modeling and simulation system is a good example. The time span for such integration is about one year or less.

Recently, *middleware based integration* is gaining its popularity. The advantage of this approach is not requiring modifications on the participating components/models/systems. In general, modifying a working system is not a favorable approach. It is a well-known fact that a software system modification itself is a source for introducing unwanted bugs in the previously working system. Additionally, a proper modification requires an in-depth knowledge of the component/model/system because of the surgical nature of the modification on an existing system. Original manufacturers often have to be involved in this modification process.

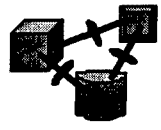
The middleware-based integration eliminates the above surgical modifications. It simply adds new middleware as needed. In the Web/IT (Information Technology) world, this approach has been widely accepted and achieved great successes. There is a recent movement to duplicate the successes in the domain of simulation integrations. A typical timeline of this approach is shorter than the Ad Hoc Interface modification approach.

6. JSB INTEGRATION INFRASTRUCTURE (JSB I2)

JSB Integration Infrastructure is not one of the above integration approaches. Only has it a similarity with the above middleware. To facilitate our further discussion, a comparison is made drawing in Figure 4.

The pros of the middleware approach have discussed in the previous section. Thus, the cons are presented here. First, the middleware interfaces are usually constructed manually, and they are created case-by-case due to the point-to-point nature of connecting two adjacent components/models/systems. Therefore, the total number of the middleware creations quickly rises (i.e., in a geometric fashion) as the number of components increases. The theoretical upper bound of the required middleware constructions is an order of N^2, where N is the number of components/models/systems to be integrated.

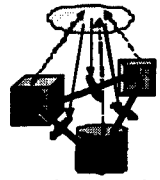
Another common aspect of the middleware approach is that they are constructed as a simple translator from one value of an originating system to another for the other receiving system. This point-to-point translation is inherent to the middleware, and it assumes one value of a system always matches with another value in the other system. Contextual aspects, which are usable and sometimes critical for a subsequent chain of translations, are rarely considered and



Middleware Integration

The Middleware Based Integration
- Pro: no modification of components
- Cons
  - Mainly manual process
  - Limited to mechanical interface translations
  - Order N^2 problem

JSB Integration Infrastructure (JSB I2)
- Ontology based integration
  - Largely automates the manual process
  - Semantics oriented interoperation
    - Not limited to mechanical translation, such as table lookup
  - Maximizes use of open standards and technologies
  - Order N problem
  - Reduces the integration time to less than 1 month



Ontology Integration

Figure 4: JSB I2 is better than the middleware based integration approach

implemented[1]. Therefore, the middleware essentially becomes a table lookup operation. If data in one system is translated to one of multiple possible values in the other system, then this approach is not applicable. A middleware approach, which is a context-free one-to-one translator, cannot handle this complex situation.

One-to-many translations are commonly performed in human language translations. Due to semantic structural differences of two languages, one representation (i.e., one meaning such as a word, a phrase or a sentence) in one language often has multiple representations in another language. Therefore, there is a low applicability of mechanical translations of human languages. Non-determinism should be resolved by a common context between two languages. It is common that machine translated text becomes a laughable object due to out-of-context usage of translated languages.

The JSB I2 approach explicitly addresses the above disadvantages of the middleware approach. First, the JSB I2 turns the N^2 implementation issue to an order N problem. Instead of manual implementation of each middleware case-by-case, a commonly ontology is implemented. The N systems are directly connected to the ontology. Thus, only the N number of connections is implemented. Figure 4 captures this concept. Then, JSB I2 automatically creates interfaces between two systems as needed. Second, the ontology maintains a common context. Thus, it is capable of resolving non-determinism of one-to-many translation cases. Moreover, it updates and maintains the common context during run-time so that it reflects the latest common context among the N participating systems. The advantage of JSB I2 is, consequently, its implementation economy by reducing the order to N^2 of the above middleware to N implementation, and its power of resolving non-determinism with the common context. It is expected that this ontology approach will significantly reduce integration of many (i.e., around 40 to 50) systems. Integration of 40 to 50 systems is a typical complexity targeted by JSB. JSB I2 is also believed to be cable of reducing the current order of 1 year integration time to a month or even shorter.

The above discussion was about the semantic sub-layer of JSB I2, which is one of the two aspects of the JSB I2. The JSB I2 has a syntactic sub-layer, and it supports the semantic sub-layer by sending and receiving a data between systems without concerning the semantic details. This sub-layer implements this context free syntactic interoperation, and this approach greatly simplifies implementation of JSB I2. A simple analogy of this syntactic integration layer is Lego piece's dimples. They allow for integration of Lego pieces without worrying about the semantic baggage associated with leg pieces, such as castle, truck, human solider Lego blocks, etc.

On top of the above syntactic interoperation, a semantic interoperation is implemented, which is the ontological portion. The current choice for the semantic representation is XML (eXtensible Markup Language). XML is also a logical choice. This choice matches with the current trend of DOD, which encourages using of XML as system interface data representations.

XML is not a just one standard of representation of data, but it comes with a family of utilities such as XSLT (eXtensible Stylesheet Language Translation), which allows for point-to-point XML translations. XML and its family of utilities really facilitate building machine understandable knowledge representations, and a direct knowledge transfer between machines. Thus, XML provides a foundation for M2M (Machine to machine). Finally, XML is also human readable.

Adopting XML as a data representation standard effectively creates multiple islands of XML'ized data language groups. XSLT easily translates one XML language to another. However, its capability is limited to a point-to-point translation. Therefore, if we rely on the standard XSLT, we essentially recreate the middleware approach discussed before. Instead, JSB I2 uses ontology to represent the common context and to translate one XML data to another. Although there has been an issue related to non-standard representation of the ontology itself, luckily the XML industry starts to develop standard ontology representations such as OWL (Web Ontology Language) [5], RDF (Resource Description Framework), etc. Therefore, the reuse of ontology will be greatly facilitated, and an incremental accumulation of ontology practically becomes possible.

Adopting the commercial standard is crucial. The commercial sector continuously improves technologies with their own investments, and JSB IF, which heavily leverages the commercial technologies, will be a beneficiary. JSB IF will benefit by this recent advance in technology as well as other technologies such as XML, etc.

7. JSB HPC APPLICATION EXAMPLES

Parallelized Common synthetic environment (CSE)

JSB CSE (Common Synthetic Environment) is a fully physics-based multi-spectrum correlated synthetic environment. The full physics-based nature is a unique feature of the CSE, but it has a drawback: it demands a high-end computational resource. Currently, in the JSB Lab, the CSE is implemented with a high end Pentium IV processor workstation, and it can execute about 10 targets in the CSE in real-time. The current single CPU architecture, thus, limits the applicability domain. An obvious answer is to parallelize the CSE, and make it run on a HPC (High Performance Computer) such as a Beowulf Cluster.

[1] Anyway, the middleware approach does not facilitate such contextual transitivity relationship. Often, middleware implementation approaches make practically impossible implementation of a contextual transitivity relationship.

The JSB program office has constructed a small Beowulf Cluster, which has three Linux computer nodes. It is designed to investigate applicability of Beowulf cluster architecture, or in general a message based parallel computer architecture for the CSE computation. Currently, each node is a Pentium IV 1.8GHz machine running Redhat Linux 7.3, and OSCAR 2.0 Beowulf software was installed to construct the Beowulf cluster.

The performance increase vs. the number of the JSB Beowulf Cluster CPUs is in the following figure.

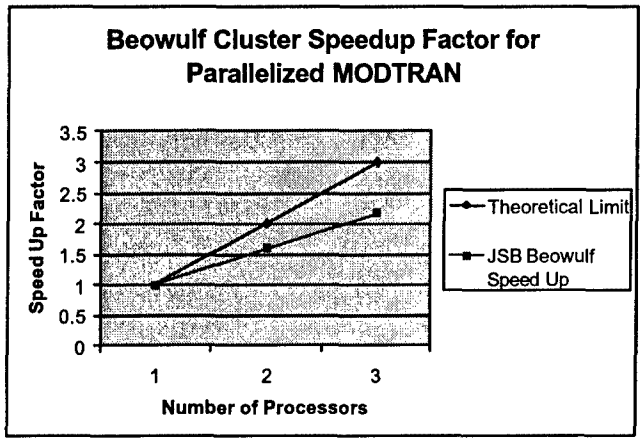


Figure 5: JSB Beowulf Cluster Speedup Factor for Parallelized MODTRAN

Making multiple computers work together to solve a common problem is a way to speed up program execution. Obviously, the more computers are involved for the same computational problem, the faster is the resultant execution speed. In theory, whenever one additional computer is added, the execution should be speeded up directly proportional to the added number of processors. In reality, this ideal scenario is seldom achieved. Overheads are always associated with a parallel computing. Thus, the rate of the performance increase is somewhat lower than the number of added processors. For example, communication latency between processors is one of the major overheads[2]. To speed up the overall performance, thus, all overheads have to be minimized.

The above graph clearly tells the story. Whenever a new computer is added, the overall execution speed is getting faster. However, the rate of the actual speed gain is less than the ideal case that can be only achieved only when absolutely no overhead is involved. In the above figure, an imaginary ideal case is labeled as "Theoretical Limit" because it can be only achieved in an ideal situation. The actual speed gain is plotted against the theoretical limit, which shows the result of parallel execution of MODTRAN running on the JSB Lab Beowulf Cluster with three processors. With the processors, the Beowulf cluster could achieve a 2.2 speed gain over that of a single CPU execution. That is, the 72% of the computing capability was contributed to achieve the overall speedup, while the 28% of that was wasted due to the overheads associated with the specific JSB Lab Beowulf cluster and the specific parallel implementation of MODTRAN algorithm

Consequently, in the world of the parallel computers, two issues are always important:

1. How to make one computing task sub-divided into multiple pieces to run simultaneously on multiple computers. That is, breaking down an algorithm into smaller sub-algorithms is the first issue. Depending on a class of problems, this is easily achieved, or sometimes it is practically impossible. If this sub-division is properly done, a significant speedup can be achieved.
2. How to minimize various overheads associated with the specific parallel computing so that the speed gain achieved by the above algorithm parallelization. The sources of overheads include data movements
3. The unwanted cost that drains the speed gain through the above algorithm parallelization is minimized. Thus, minimizing the overheads becomes the name of the game, such as communication overheads associated when moving data crossing CPU/machine boundaries. There are also other sources of overheads: protocol conversion, interconnection topology (number of hops between CPUs), etc.

It is obvious that the real gain of the parallel computer mainly comes from the algorithm parallelization. Then the gain is decreased by various overheads associated with a particular parallel computer implementation, such as the JSB Beowulf Cluster.

JSB photo realistic Visualization Requirement

People often say that a higher resolution graphics is better. However, if they are asked what the quantitative specification is, then they are hardly able to describe the specifics. At best, a term, "photo realistic visualization" is commonly used, which is a subjective measure. Thus, the JSB program office has developed a specific "photo-realistic visualization" requirement for the JSB applications.

The high-end graphic resolution (i.e., so called photo-realistic graphic resolution) for JSB applications is computed based on the AFRL (Air Force Research Lab) research result -- our human eyes are capable of detection

[2] A special class of problem that does not require a communication while computing a solution collectively (i.e., in parallel). In this case, this type of communication overhead is so small (or sometimes practically none) that the speedup factor is directly proportional to the number of the CPUs added to the specific problem solving.

differences as small as 0.5 arc minutes[3] under a high contrast situation such as a target against a bright sky background [6]. Then to provide a full 360-degree view including an overhead view, we need 467M pixels:

(360*60*2)*(90*60*2) = 467M pixels. (1)

Rendering graphic images with the above level of resolution at a given moment is critical in order not to wash out the important visual cues used by a human pilot under a real air-to-air combat situation. This requirement is more stringent than that of many cockpit trainers currently available. To provide continuous motion pictures, the above 467M-pixel scene has to be updated 30 times per second or more. Then the required number of pixels to be processed by a graphic computer per second is 14,010M pixels:

467M*30 = 14,010M pixels per second (2)

This numb er is roughly equivalent to 16 graphic pipes of the latest graphics hardware technology:

(14,010M pixels/sec) / (896M pixels/sec/pipe) = about 16 pipes. (3)

Therefore, a 16 graphic pipe system is ideal for JSB applications.

However, the above requirement can be relaxed. A full 360-degree view does not have to be rendered all the time because of the following reasons:

1. A human pilot cannot see a full 360-degree view at a time.
2. All visible areas to a human pilot do not require to be rendered at the maximum resolution described above.

That is, only the front (i.e., 120° x 90°) visual area requires the highest resolution graphics, while the other area may be able to use a lower 1 arc minute resolution, which is equivalent to 20/20 vision. This relaxation results in 5 graphic pipes for the front, and 1 graphic pipe to cover the peripheral vision as follows:

1. Detailed 120° x 90° frontal area → about 5 graphic pipes are required
2. Less detailed area (180° - 120°) x 90° scene → about 1 graphic pipe is enough

To support the above approach, a device that can continuously track the frontal eyesight area has to be augmented.

8. SUMMARIES AND CONCLUSION

The JSB is a noble concept that significantly facilitates design, analysis, development, acquisition, training, and operations of future mission oriented C2/weapon systems, which are essentially a complex system of systems such as C2 constellation. Realizing the JSB's noble concept requires leveraging all existing assets (i.e., simulation systems, communication infrastructures, organizations, etc) rather than building from scratch. Thus, the JSB needs to have proper and adequate capability and process that allows for quick integration of existing assets. Many of them are directly associated with human organization and programmatic aspects, but a new technical break-through is crucial. The noble concept requires significant technological breakthroughs in many areas including integration framework, modeling and simulations, computational physics, phenomenology, human factor/decision making behaviors, distributed computing, network centric operations, multi-level securities, dynamic VV&A, and parallel computations. Surely, many more areas have not even been mentioned here and many more will be uncovered. As the JSB core technologies, architectures and processes mature, the JSB will start to reach the full scope of the vision while significantly impacting everyday operations of USAF and other DOD services. Some of the technologies described in this paper are essential as a necessary step toward the "real" JSB. The problem space is huge, and our knowledge is limited. Our current efforts will surely bear the fruits while providing much needed valuable data moving forward to the "real" JSB in future.

[3] The Snellen eye chart used by most optometrists tests a healthy human visual acuity at 1 minute of arc, which is commonly known as 20/20 vision.

References

1. USAF, *Joint Synthetic Battlespace For Simulation Based Acquisition*, JSB Concept of Operation (CONOPS), 2001.
2. Kwak, S.D., "A Multiple Paradigm Behavior Architecture: COREBA (Cognition Oriented Emergent Behavior Architecture)", *Proceedings of 1998 Fall Simulation Interoperability Workshop*, SISO, Orlando, FL, Sept. 14-18, 1998.
3. Kwak, S.D., Andrew, E.B., "Technical Challenges for Joint Synthetic Battlespace (JSB)", *Proceedings of 2002 Fall Simulation Interoperability Workshop*, SISO, Orlando, FL, Sept. 9-13, 2002
4. Moore, G.E., "Cramming more components onto integrated circuits", *Electronics*, Volume 38, Number 8, April 19, 1965.
5. http://www.w3.org/TR/2003/WD-owl-features-20030331/, W3C, March 2003
6. Snow, M.P., Jackson, T.W., Meyer, F.M., Reising, J.M., Hopper, D.G., The AMLCD cockpit: promise and payoffs, *Cockpit Displays VI: Displays for Defense Applications* (pp. 103-114). Bellingham, WA: International Society of Optical Engineers (SPIE), March 1999

9. DISCLAIMER

The author's affiliation with The MITRE Corporation is provided for identification purposes only, and is not intended to convey or imply MITRE's concurrence with, or support for, the positions, opinions or viewpoints expressed by the author.